Introduction to Operating Systems

Tom Anderson
Adam Lerner
Will Scott
How This Course Fits in the UW CSE Curriculum

• CSE 333: Systems Programming
  – Project experience in C/C++
  – How to use the operating system interface
• CSE 451: Operating Systems
  – How to make a single computer work reliably
  – How an operating system works internally
• CSE 452: Distributed Systems (spring 2013)
  – How to make a set of computers work reliably, despite failures of some nodes
Project: Pintos

• Build an operating system
  – That can boot on PC hardware
  – Run a web server (and other apps)
• We give you some basic building blocks
  – Four assignments, that build on each other
    • Threads, user programs, virtual memory, networking
  – Work in groups of 2-3
• First assignment due two weeks from Thursday
Problem Sets

• Four assignments spread over quarter
  – Practice for exams
  – Done individually

• First assignment, due a week from Thursday
  – Build a shell, with pipes: ls | wc
  – Should be review from 333
  – See Chapter 3.1-3.4 for a review
  – Posted online
Main Points (for today)

• Operating system definition
  – Software to manage a computer’s resources for its users and applications

• OS challenges
  – Reliability, security, responsiveness, portability, ...

• OS history
  – How are OS X, Windows 7, and Linux related?
What is an operating system?

- Software to manage a computer’s resources for its users and applications.
Operating System Roles

• **Referee:**
  – Resource allocation among users, applications
  – Isolation of different users, applications from each other
  – Communication between users, applications

• **Illusionist**
  – Each application appears to have the entire machine to itself
  – Infinite number of processors, (near) infinite amount of memory, reliable storage, reliable network transport

• **Glue**
  – Libraries, user interface widgets, …
Thought Question

• What do you need from hardware to be able to:
  – Isolate different applications from each other?
  – Isolate different users from accessing each others files?
Example: web service

- How does the server manage many simultaneous client requests?
- How do we keep the client safe from spyware embedded in scripts on a web site?
- How do we keep updates to the web site consistent?
OS Challenges

- **Reliability**
  - Does the system do what it was designed to do?
  - **Availability**
    - What portion of the time is the system working?
    - Mean Time To Failure (MTTF), Mean Time to Repair

- **Security**
  - Can the system be compromised by an attacker?
  - **Privacy**
    - Data is accessible only to authorized users

- Both require very careful design and code
OS Challenges

• Portability
  – For programs:
    • Application programming interface (API)
    • Abstract machine interface
  – For the operating system
    • Hardware abstraction layer
    • Pintos provides hardware-specific OS kernel routines
OS Challenges

• Performance
  – Latency/response time
    • How long does an operation take to complete?
  – Throughput
    • How many operations can be done per unit of time?
  – Overhead
    • How much extra work is done by the OS?
  – Fairness
    • How equal is the performance received by different users?
  – Predictability
    • How consistent is the performance over time?
OS History

- MVS (60's)
  - MS/DOS (70's)
    - Windows (80's)
      - Windows Mobile NT
      - Windows (90's)
      - Windows 8 (2012)
    - VMS (70's)
    - FreeBSD (80's)
    - Linux (90's - present)
    - Android
    - MacOS X
  - Windows (80's)
  - FreeBSD (80's)
  - Linux (90's - present)
  - VMware
  - Mach (80's)
  - Next

- Multics (60's)
  - FreeBSD (80's)
  - Mach (80's)
  - FreeBSD (80's)
  - Linux (90's - present)
  - VMware

- UNIX (70's)
  - FreeBSD (80's)
  - Linux (90's - present)

- Windows (80's)
  - FreeBSD (80's)
  - Linux (90's - present)
  - VMware
  - Mach (80's)
  - FreeBSD (80's)
  - Linux (90's - present)

Influence

Descendant
## Computer Performance Over Time

<table>
<thead>
<tr>
<th></th>
<th>1981</th>
<th>1996</th>
<th>2011</th>
<th>factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIPS</td>
<td>1</td>
<td>300</td>
<td>10000</td>
<td>10K</td>
</tr>
<tr>
<td>MIPS/$</td>
<td>$100K</td>
<td>$30</td>
<td>$0.50</td>
<td>200K</td>
</tr>
<tr>
<td>DRAM</td>
<td>128KB</td>
<td>128MB</td>
<td>10GB</td>
<td>100K</td>
</tr>
<tr>
<td>Disk</td>
<td>10MB</td>
<td>4GB</td>
<td>1TB</td>
<td>100K</td>
</tr>
<tr>
<td>Home Internet</td>
<td>9.6 Kbps</td>
<td>256 Kbps</td>
<td>5 Mbps</td>
<td>500</td>
</tr>
<tr>
<td>LAN network</td>
<td>3 Mbps (shared)</td>
<td>10 Mbps</td>
<td>1 Gbps</td>
<td>300</td>
</tr>
<tr>
<td>Users per machine</td>
<td>100</td>
<td>1</td>
<td>&lt;&lt;1</td>
<td>100+</td>
</tr>
</tbody>
</table>
Early Operating Systems: Computers Very Expensive

• One application at a time
  – Had complete control of hardware
  – OS was runtime library
  – Users would stand in line to use the computer

• Batch systems
  – Keep CPU busy by having a queue of jobs
  – OS would load next job while current one runs
  – Users would submit jobs, and wait, and wait, and
Time-Sharing Operating Systems: Computers and People Expensive

- Multiple users on computer at same time
  - Multiprogramming: run multiple programs at same time
  - Interactive performance: try to complete everyone’s tasks quickly
  - As computers became cheaper, more important to optimize for user time, not computer time
Today’s Operating Systems: Computers Cheap

- Smartphones
- Embedded systems
- Web servers
- Laptops
- Tablets
- Virtual machines
- …
Tomorrow’s Operating Systems

- Giant-scale data centers
- Increasing numbers of processors per computer
- Increasing numbers of computers per user
- Very large scale storage
Bonus Thought Question

• How should an operating system allocate processing time between competing uses?
  – Give the CPU to the first to arrive?
  – To the one that needs the least resources to complete? To the one that needs the most resources?
  – What if you need to allocate memory?
  – Disk?
Textbook

• Lazowska, Spring 2012: “The text is quite sophisticated. You won't get it all on the first pass. The right approach is to [read each chapter before class and] re-read each chapter once we've covered the corresponding material... more of it will make sense then. Don't save this re-reading until right before the mid-term or final – keep up.”